تکلیف هشت - مهندسی نرم افزار 1

بررسی و بهبود کیفیت طراحی

محمد امانلو 810100084 - شهزاد ممیز 810100272 - گلبو رشیدی 810100148

Bloaters:

1. Long Method:

a. **Extract Method: You have a code fragment that can be grouped together. Move this code to a separate new method (or function) and replace the old code with a call to the method.**

b. **Replace Temp with Query: You place the result of an expression in a local variable for later use in your code.**

c. **Introduce Parameter Object: Your methods contain a repeating group of parameters. Replace these parameters with an object.**

d. **Preserve Whole Object: You get several values from an object and then pass them as parameters to a method. Instead, try passing the whole object.**

e. **Replace Method with Method Object: You have a long method in which the local variables are so intertwined that you can't apply *Extract Method*. Transform the method into a separate class so that the local variables become fields of the class. Then you can split the method into several methods within the same class.**

f. **Decompose Conditional: You have a complex conditional (`if-then/else` or `switch`). Decompose the complicated parts of the conditional into separate methods: the condition, `then` and `else`.**

2. Primitive Obsession: Use of primitives instead of small objects for simple tasks (such as currency, ranges, special strings for phone numbers, etc.)

a. **Introduce Parameter Object:** problem: Your methods contain a repeating group of parameters. solution: Replace these parameters with an object.

b. **Preserve Whole Object**: problem: You get several values from an object and then pass them as parameters to a method. solution: Instead, try passing the whole object.

c. **Replace Type Code with Class:** for example, create a new class for blood type instead of defining all of them(A,O,AB,..) and use bool.

d. **Replace Type Code with Subclasses:** for example for animal we create subclasses cat and dog.
e. **Replace Type Code with State/Strategy.**
f. **Replace Array with Object:** when we have array a=[15, “ali”, “yazd”] we use object instead.

3. **Data Clumps: If you want to make sure whether or not some data is a data clump, just delete one of the data values and see whether the other values still make sense.**
   a. **Extract Class**
   b. **Introduce Parameter Object:** problem: Your methods contain a repeating group of parameters. Solution: Replace these parameters with an object.
   c. **Preserve Whole Object:** You get several values from an object and then pass them as parameters to a method. Instead, try passing the whole object.

4. **Large Class: A class contains many fields/methods/lines of code.**
   a. **Extract Class helps if part of the behavior of the large class can be spun off into a separate component.**
   b. **Extract Subclass helps if part of the behavior of the large class can be implemented in different ways or is used in rare cases.**
   c. **Extract Interface helps if it’s necessary to have a list of the operations and behaviors that the client can use.**

5. **Long Parameter List**

   a. **Replace Parameter with Method Call: Instead of passing the value through a parameter, try placing a query call inside the method body.**
   b. **Preserve Whole Object:**problem: You get several values from an object and then pass them as parameters to a method. solution: Instead, try passing the whole object.

c. Introduce Parameter Object: Your methods contain a repeating group of parameters. Replace these parameters with an object.

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
d06c0c78613edf09eb0782b7fbbb3b6556e37b92	Replace Method with method object	695f1e1df69854bf2f6a25acc5804528d092b3e7	در کلاس OrderHandler ما توابع زیاد و طولانی ای را برای validate کردن request های خود داریم. برای رفع این مشکل یک کلاس ValidateRq باید به اضافه کنیم بعنوان Method object.
7cc7213e031000c6b96f55addf91960ddf387177	Extract Method	695f1e1df69854bf2f6a25acc5804528d092b3e7	یک متد بزرگ برای validate کردن داشتیم که آن را به قسمت های کوچکتر شکستیم: - validate security - validate order - validate broker - validate shareholder
7cc7213e031000c6b96f55addf91960ddf387177	Extract Method	559809d378c74139fc16cd58a8bf9eb4b9e42192	انجام تغییرات اکتیو کردن stop limit order در تابع handle enter order انجام میشد. بررسی شرایط و اعمال تغییرات به تابع دیگری منتقل شد.
6fc675ee324ccb698563a0e4140dbf94c45750bf	Extract Method	695f1e1df69854bf2f6a25acc5804528d092b3e7	باز آرایی تابع **publishOutcome**:
6fc675ee324ccb698563a0e4140dbf94c45750bf	Extract Method	df9b9c2cf81e5f8d1e441d76cdb4b1893a23c916	refactoring **handleEnterOrder**: Extract validation and processing logic into separate methods.

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
f602fe134ae9dcc65949051eb181fe301c7dfe9d	Extract Method	695f1e1df69854bf2f6a25acc5804528d092b3e7	**execInactiveStopLimitOrders** Method Refactoring
f602fe134ae9dcc65949051eb181fe301c7dfe9d	Extract Method	695f1e1df69854bf2f6a25acc5804528d092b3e7	execInactiveStopLimitOrdersAuction Method Refactoring
f602fe134ae9dcc65949051eb181fe301c7dfe9d	Extract Method	581309f9b39332e781bced6e08d665e86b6744be	handleChangeMatchStateRq Method Refactoring: Extract match state change and event publishing logic into separate methods.
b74379de49e0e093d199a8f35ddec1fca38b984b	Extract Methods	95e47e688411e4443382758f0dd65a77aa772763	Simplifing the **inactiveOrderQueuesBefore** Method: Breking down into smaller methods to handle the comparison logic
6fc675ee324ccb698563a0e4140dbf94c45750bf	Long method, Larg Class	a5c2272de5c46326420741114ed5cffcc5bd2aea	Refactored Constructors: Simplifing constructor overloads to reduce redundancy and maintain consistency. Used a primary constructor and called it from secondary constructors.
9815d92d38a22e4629802c0e19d406e9546ab0ac	Long method, Larg Class	a274170c579d8f3c6108813c48dab274faa07c88	Refactoring to eliminate code duplication by extracting the common enqueuing logic into a new method
559809d378c74139fc16cd58a8bf9eb4b9e42192	Extract Methods	1b86570db921b19b727f326d7b6ab173a23be4d3	در ابتدا ما تابع processExecutableOrder را داشتیم و تابع PublishTrades را از آن

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
			جدا کردیم.
93511c0a49339421a6a7ed8a7c8a7cdfeb4b6ba5	Extract Methods Long method(Replace Method with Method Object)	df9b9c2cf81e5f8d1e441d76cdb4b1893a23c916	clean publishing logic in OrderHandler
10a780a35d77747ffd8f6baf786fdda0dae93424	Decompose Conditional	c923e7001fed36a19de75a82d95b55092425aed1	we reduced if condition length. and we added new method named: “isDeleteStopLimitInAuction”-
cf3252a41aad82cbd1797cbdd232879f89d8c80b	Decompose Conditional	3ee48313ca6011949106c55d18caced7925322a3	در قسمتی نیز if کد کثیف داشتیم که decomposition انجام دادیم.
cf3252a41aad82cbd1797cbdd232879f89d8c80b	extract method	3ee48313ca6011949106c55d18caced7925322a3	ساختن order جدید که قبلا در new order بود و extract method کردیم.
29b63a0387a90ffec81bc953f9f6113420d764f0	Extract Method (Long Method: Decompose Conditional)	1ed82650e48a3e9761954f46b9794c8521fe5102	متود های activateSellStopLimitOrders و activateBuyStopLimitOrders منطق شرطی پیچیده ایی دارند که برای ساده سازی آن ها را در متود جدید قرار میدهیم.
29b63a0387a90ffec81bc953f9f6113420d764f0	Extract Method (Long Method)	bd0b9d1e9f8be54f6a34f51ca6068148cc75085c	در متود totalSellQuantityByShareholder لاجیک حساب کردن مجموع را تغییر داده یک متود جدید از این متود استخراج کردیم.

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
29b63a0387a90ffec81bc953f9f6113420d764f0	Long Method	c1cb2ba89aded508510f8abf285e08cda97deb51	متود های snapshot و snapshotWithQuantity duplicated code دارند. که قسمت مشترک را یک متود جدید میکنیم.
29b63a0387a90ffec81bc953f9f6113420d764f0	Large Class	907bad8d84998e78c93d3a70e3f73fe2d033ec05	متود queuesBefore(StopLimitOrder order) اضافی است و نیازی به آن نیست.
907bad8d84998e78c93d3a70e3f73fe2d033ec05	extract method	84ae5a406f82b6dbcee9d1de0b027e4b0de7bcb9	متد process order را extract کردیم زیرا در return تابع کد کثیفی زده شده بود و آن را به تابع تبدیل کردیم.
84ae5a406f82b6dbcee9d1de0b027e4b0de7bcb9	extract method	ec033a277c498aba1fd3a192667f0cd59119315a	تابع find order را ایجاد کردیم و از تابع قبلی اش extract کردیم
ec033a277c498aba1fd3a192667f0cd59119315a	extract method	8ac144f21751cece36810c0821806e7249f60207	تابع handleBuyDeletedOrderCredit را اضافه کردیم و عملیاتی که انجام می داد را از تابع قبلی از جدا (extract) کردیم
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateInvalidUpdatePeakSize

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateNonIcebergHavingPeakSize
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateUpdateStopPriceForNonStopLimit
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateZeroStopPriceForStopLimit
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateStopLimitHaveMEQ
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateStopLimitBeIceberg

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateUpdateActiveStopLimit
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateUpdateMEQ
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateUpdateStopLimit
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validatePeakSize
8ac144f21751cece36810c0821806e7249f60207	extract method	bd547e4469f0c53536e1e4f801f1961b72ebfb6e	validateStopLimitBeIceberg

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
bd547e4469f0c53536e1e4f801f1961b72ebfb6e	extract method	3860e093261f3bd50d3c8d6d9f890a152db6fe13	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: validateUpdateOrder
a5fc5087777d997fabaa1401fa6919c241a1bb3b	Long Method - Decompose Conditional	b11691ae5b69e9a44754e82de3d5b5d0c5d981b3	در کاندیشنی که داشتیم، تابع isLosesPriority را تعریف و جایگزین کد های موجود کردیم.
5d9d93e8bf6421c3b444e558fdfba84326941218	extract method	fa56938a79e97dea679471992140a6376cd249f8	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: executeUpdatedOrder
5d9d93e8bf6421c3b444e558fdfba84326941218	extract method	fa56938a79e97dea679471992140a6376cd249f8	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: executeActiveOrder
5d9d93e8bf6421c3b444e558fdfba84326941218	extract method	fa56938a79e97dea679471992140a6376cd249f8	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: removePrevOrder

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
5d9d93e8bf6421c3b444e558fdfba84326941218	extract method	fa56938a79e97dea679471992140a6376cd249f8	برای تابع زیر متد اکسترکت انجام دادیم و قسمتی از فانکشنالیتی را به تابع زیر منتقل کردیم: enqueueUpdatedOrder
3f959d1f81599688a7cfaa4803022d4e47c2c580	extract method	99cf2adf802bfd46572afc45fa8ebfe3fc963baf	برای opening price ، اکسترکت متد انجام دادیم
99cf2adf802bfd46572afc45fa8ebfe3fc963baf	extract method	a58dddde058fe9421ec3d877f7b0bfae0899c039	برای findoverallQuantity اکسترکت متد انجام دادیم. و دو تابع findBuyOrdersToTrade و findSellOrdersToTrade را اضافه کردیم.
a58dddde058fe9421ec3d877f7b0bfae0899c039	extract method	e936f1f35511f61d9ff33433dbb80bd3b73a207e	برای findDistanceToLastTradePrice اکسترت متد انجام دادیم.
e936f1f35511f61d9ff33433dbb80bd3b73a207e	extract method	2755eef1893e9b5769d07ac8a9d9fdc4287e3480	برای updateMinElement_and updateMinDistance اکسترکت متد انجام دادیم.

توضیحات (در صورت نیاز)	شناسه کامیت بازآرایی‌شده	شرح مختصر بوی بد	شناسه کامیت حاوی بوی بد
we did “refactoring findCandidatePrices method” we encapsulated price and quantity in the “ priceQuanitity” class.	629d23ee712800ca2106dbc85936306b205c06de	Primitive Obsession	df6aa312ca1fd4688d56f1bd6923934d25b3a0da
Extracting the “calculatePriceQuantities” and “getBestOpenPrices” methods to break down the “findCandidatePrices” method.	629d23ee712800ca2106dbc85936306b205c06de	long method - extract method	df6aa312ca1fd4688d56f1bd6923934d25b3a0da
Extracting Methods for Gathering Prices: Extracte gatherAllOrderPrices, gatherPricesFromQueue, and setIndicativeOpening Price methods to break down the updateIndicativeOpeningPrice method.	a7a95ca0e4e9e1cac54784251e44e7cf260effce	Long Method - extract method	629d23ee712800ca2106dbc85936306b205c06de

توضیحات (در صورت نیاز)	شناسه کامیت بازآرایی‌شده	شرح مختصر بوی بد	شناسه کامیت حاوی بوی بد
Managed prices within a collection, enhancing readability and maintainability.	a7a95ca0e4e9e1cac54784251e44e7cf260effce	Primitive Obsession	629d23ee712800ca2106dbc85936306b205c06de
refactor auctionAddToQueue method: Managing redundant data access within encapsulated methods, reducing code duplication, and improving clarity.	a7a95ca0e4e9e1cac54784251e44e7cf260effce	Data Clumps	629d23ee712800ca2106dbc85936306b205c06de
refactoring execute and auctionExecute: Extracting methods to handle specific tasks within the execute and auctionExecute methods.	a9f23c88ef5701c7da1c306b1abb39944c18f88f	extract method	a7a95ca0e4e9e1cac54784251e44e7cf260effce
refactoring execute and auctionExecute: Managing quantities and prices within the encapsulated methods.	a9f23c88ef5701c7da1c306b1abb39944c18f88f	Primitive Obsession	f888ef948ed4337b9c91be9f984ccba8dc5f816c

توضیحات (در صورت نیاز)	شناسه کامیت بازآرایی‌شده	شرح مختصر بوی بد	شناسه کامیت حاوی بوی بد
refactoring execute and auctionExecute Consolidating the check for matching outcomes into a single method isMatchingOutcomeNotEnough	a9f23c88ef5701c7da1c306b1abb39944c18f88f	Refactored Conditions	f888ef948ed4337b9c91be9f984ccba8dc5f816c
فانکشن check position if را اضافه کردیم. و به جای not ، آن را قرار دادیم و کد تمیز تر شد	5d9d93e8bf6421c3b444e558fdfba84326941218	extract class	b11691ae5b69e9a44754e82de3d5b5d0c5d981b3
refactor auctionMatch method Extracting methods to handle specific tasks within the auctionMatch method.	64f6a0de9a9f6c992642ee2c51522f05439797cb	extract method	a9f23c88ef5701c7da1c306b1abb39944c18f88f
refactor auctionMatch method Managing quantities and prices within the encapsulated methods.	64f6a0de9a9f6c992642ee2c51522f05439797cb	Primitive Obsession	a9f23c88ef5701c7da1c306b1abb39944c18f88f

Object-Orientation Abusers:

1. **Alternative Classes with Different Interfaces:** Two classes perform identical functions but have different method names.

2. **Refused Bequest**: If a subclass uses only some of the methods and properties inherited from its parents, it happens.
    a. **Replace Inheritance with Delegation.**
    b. **Extract Superclass**

3. **Switch Statements: You have a complex `switch` operator or sequence of `if` statements**
    a. **Extract Method**
    b. **Move Method.**
    c. **Replace Type Code with Subclasses**
    d. **Replace Type Code with State/Strategy**
    e. **Replace Conditional with Polymorphism**
    f. **Replace Parameter with Explicit Methods**
    g. **Introduce Null Object**

4. **Temporary Field: Temporary fields get their values (and thus are needed by objects) only under certain circumstances. Outside of these circumstances, they're empty.**
    a. **Extract Class.**
    b. **Replace Method with Method Object.**
    c. **Introduce Null Object**

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
d06c0c78613edf09eb0782b7fbbb3b6556e37b92	Refused Bequest	f8dd9d95b48dab0ba15d35972b46d339a70e756a	یکسری کلاس به نام های زیر داشتیم. یک کلاس Request ساختیم تا کلاسهای زیر از آن ارث بری کنند. ChangeMatchStateRq, DeleteOrderRq, EnterOrderRq
df6aa312ca1fd4688d56f1bd6923934d25b3a0da	Temporary Field - extract class	629d23ee712800ca2106dbc85936306b205c06de	we did "refactoring findCandidatePrices method" Temporary Field: Managed temporary

data within the scope of the method through helper methods.			
Temporary Field: Managed temporary data within the scope of the method through helper methods, improving clarity and reducing complexity.	a7a95ca0e4e9e1cac54784251e44e7cf260effce	Temporary Field - Extract Class.	629d23ee712800ca2106dbc85936306b205c06de
we did refactor the match method in Mather Extracting Methods from the match and simplify it.	f888ef948ed4337b9c91be9f984ccba8dc5f816c	Extract Methods	a7a95ca0e4e9e1cac54784251e44e7cf260effce
refactor auctionAddToQueue method was done. Long Method: Extracted checkBuyerCreditIfNeeded, checkSellerPositionsIfNeeded, checkBuyerCredit, checkSellerPositions, and enqueueOrderAndUpdatePrice methods to break down the auctionAddToQueue method.	f888ef948ed4337b9c91be9f984ccba8dc5f816c	extract method	be458adf2d56b63481c6c2c7068375ca1b470799
refactor auctionAddToQueue method Primitive Obsession: Managed credit and position checks within	f888ef948ed4337b9c91be9f984ccba8dc5f816c	Primitive Obsession	be458adf2d56b63481c6c2c7068375ca1b470799

encapsulated methods, enhancing readability and maintainability.			

Change Preventers

1. **Divergent Change: You find yourself having to change many unrelated methods when you make changes to a class. For example, when adding a new product type you have to change the methods for finding, displaying, and ordering products.**

    a.Extract Class: Split up the behavior of the class via **Extract Class**.

    b.Extract Superclass and **Extract Subclass:** If different classes have the same behavior, you may want to combine the classes through inheritance (**Extract Superclass** and **Extract Subclass**).

2. **Parallel Inheritance Hierarchies: Whenever you create a subclass for a class, you find yourself needing to create a subclass for another class.**
    a. **Move Method**
    b. **Move Field.**

3. **Shotgun Surgery: Making any modifications requires that you make many small changes to many different classes.**
    c. **Move Method**
    d. **Move Field.**
    e. **inline class**

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
1b86570db921b19b727f326d7b6ab173a23be4d3	Extract Class	c32b33f7d248720d7b07d950d9a78e9b15e078a1	یک کلاس validate اضافه کردیم. که همه ی validation های delete و enterNewOrder را انجام دهد

Dispensables

1. Comments: A method is filled with explanatory comments.

a.Extract Variable: Problem: You have an expression that's hard to understand. Solution:Place the result of the expression or its parts in separate

b. Extract Method

c. Rename Method

d. Introduce Assertion: Problem: For a portion of code to work correctly, certain conditions or values must be true. **Solution:** Replace these assumptions with specific assertion checks.

2. Data Class: A data class refers to a class that contains only fields and does not have functions.

Encapsulate Field

Encapsulate Collection

Move Method

Extract Method

Remove Setting Method

Hide Method

3. Lazy Class: Understanding and maintaining classes always costs time and money. So if a class doesn't do enough to earn your attention, it should be deleted.

a. **Inline Class: Move all features from one class to another one.**

b. **Collapse Hierarchy:** Problem: **You have a class hierarchy in which a subclass is practically the same as its superclass.** Solution: **Merge the subclass and superclass.**

4. Duplicate Code: Two code fragments look almost identical.

a. **Extract Method**
b. **Pull Up Field**
c. **Pull Up Constructor Body.**
d. **Form Template Method.**
e. **Substitute Algorithm.**
f. **Extract Superclass**
g. **Extract Class**
h. **Consolidate Conditional Expression:** Problem: You have multiple conditionals that lead to the same result or action. Solution: Consolidate all these conditionals in a single expression.
i. **Consolidate Duplicate Conditional Fragments:** Problem: Identical code can be found in all branches of a conditional. Solution: Move the code outside of the conditional.

5. Dead Code: A variable, parameter, field, method or class is no longer used (usually because it's obsolete).

a. **IDE.**
b. **Inline Class**
c. **Collapse Hierarchy**
d. **Remove Parameter.**

6. Speculative Generality: There's an unused class, method, field or parameter.

a. **Collapse Hierarchy**
b. **Inline Class**
c. **Inline Method**
d. **Remove Parameter**

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)

تابع decreaseBuyCredit را اضافه کردیم. و کد های تکراری مربوطه را ک در چند جای کد وجود داشت را حذف و همه را به تابع تبدیل کردیم.	1ef8b626ddb7794744a9c16849730fe35c269174	Duplicate Code- Form Template Method	
کد تکراری داشتیم که آن را حذف کردیم. و تابع updateIndicativeOpeningPrice در چندین تا از if های پشت سر هم استفاده شده بود آن را به بالای پروژه منتقل کردیم.	f78db8582bcbf187fef71f429769ef110bb7dc26	Duplicate Code - Consolidate Duplicate Conditional Fragments:	fa56938a79e97dea679471992140a6376cd249f8
کد تکراری داشتیم که آن را حذف کردیم. و if هایی که داشتیم را با هم مرج کردیم.	62e7674b3d04293dbf8c4889e3eb77b3b2f18c79	Duplicate Code - Consolidate Duplicate Conditional Fragments:	f78db8582bcbf187fef71f429769ef110bb7dc26
فیلدی به اسم max داشتیم که به خوبی تبیین کننده ی محتوای متغیر نبود و اسم آن را به sellQueueSize تغییر دادیم.	3f959d1f81599688a7cfaa4803022d4e47c2c580	rename field	62e7674b3d04293dbf8c4889e3eb77b3b2f18c79
قسمتی از کد بود که استفاده نمیکردیم و اضافی بود آن را حذف کردیم.	877f18a19e8950dd04b596e09959b6b581b8e982	Dead Code - remove	2755eef1893e9b5769d07ac8a9d9fdc4287e3480
فیلدی که min بود را اسمش ر به minPrice تغییر دادیم چون خیلی گویا نبود.	04dfb5386d19eaa9b85c525ee328d9cf58dbd018	rename parameter	877f18a19e8950dd04b596e09959b6b581b8e982
refactoring execute and auctionExecute Duplicate Code: Extracted common logic for updating trade positions into the updateLastTradePric	a9f23c88ef5701c7da1c306b1abb39944c18f88f	Duplicate Code	f888ef948ed4337b9c91be9f984ccba8dc5f816c

eAndPositions method.			
متدpublishOutcome را تغییر دادیم به: publishDelete publishEnterRq	93511c0a49339421a6a7ed8a7c8a7cdfeb4b6ba5	renaming	1b86570db921b19b727f326d7b6ab173a23be4d3
drop extra lines and comments	cf3252a41aad82cbd1797cbdd232879f89d8c80b	Comments, Dead Code	fd35999679e45403daaca13f049fef25d5bccb72
we deleted duplicated conditions and put “if”s together.	ec033a277c498aba1fd3a192667f0cd59119315a	Duplicate Code - Consolidate Duplicate Conditional Fragments:	84ae5a406f82b6dbcee9d1de0b027e4b0de7bcb9
refactoring findCandidatePrices method انجام دادیم. در واقع چندین عملیات مختلف انجام دادیم : we encapsulated price and quantity in the “ priceQuanitity” class.	629d23ee712800ca2106dbc85936306b205c06de	Encapsulate Collection	df6aa312ca1fd4688d56f1bd6923934d25b3a0da
we deleted an unused function at the end of matchResult	fd35999679e45403daaca13f049fef25d5bccb72	dead code - Remove Parameter	df9b9c2cf81e5f8d1e441d76cdb4b1893a23c916
refactor auctionMatch method Duplicate Code: Extracted common logic for trade creation, credit adjustment, and	64f6a0de9a9f6c992642ee2c515220543979 7cb	Duplicate Code	a9f23c88ef5701c7da1c306b1abb39944c18f88f

updating order quantities into separate methods.			
handleBuyOrderCredit	43fe0eda58bd5d9b2ab65fce3cd1bb0f9d3761ef	Duplicate Code	3860e093261f3bd50d3c8d6d9f890a152db6fe13
کد هایی که از آن استفاده نمی کردیم را حذف کردیم.	1f85b95bda14c3e35aea45a4b68ada4cbe98499a	dead code - Remove Parameter	d3766958a754d8d6a2cf5bd912858ba95ba33d95

Couplers

1. Feature Envy: A method that accesses the data of another object more than its own data.

- **a. Move Method: If a method clearly should be moved to another place, use the Move Method.**
- **b. Extract Method: If only part of a method accesses the data of another object, use the Extract Method to move the part in question.**

2. Inappropriate Intimacy: One class uses the internal fields and methods of another class.

- **a. Move Method**
- **b. Move Field**
- **c. Extract Class**
- **d. Hide Delegate: The client gets object B from a field or method of object A. Then the client calls a method of object B. solution: Create a new method in class A that delegates the call to object B. Now the client doesn't know about, or depend on, class B.**
- **e. Change Bidirectional Association to Unidirectional.**
- **f. Replace Delegation with Inheritance.**

3. Incomplete Library Class

4. Message Chains: In code, you see a series of calls resembling $a->b()->c()->d()

- **a.Hide Delegate.**
- **b. Extract Method**
- **c. Move Method.**

5. Middle Man: If a class performs only one action, delegating work to another class, why does it exist at all?

- **a. Remove Middle Man**

شناسه کامیت حاوی بوی بد	شرح مختصر بوی بد	شناسه کامیت بازآرایی‌شده	توضیحات (در صورت نیاز)
8ac144f21751cece36810c0821806e7249f60207	Feature Envy- Move Method	121624e1a6a99e6c76cccff6a7aebbdafd330bcd	validateUpdateOrderRq را اضافه کردیم.

Design Patterns:

1) Factory Method:

we have applied this design pattern to our code. this is the SHA:
d3766958a754d8d6a2cf5bd912858ba95ba33d95
Factory Method is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.
Use Case: Creating instances of various Order types (IcebergOrder, StopLimitOrder, etc.).

Implementation: You could have an OrderFactory interface with a method createOrder(). Concrete factories would implement this method to create specific types of orders.

First, we define an interface for creating orders.

Next, we create concrete factory classes for each type of Order.

Then we modify your OrderHandler (or wherever orders are created) to use the factory.

2) Chain of Responsibility

we have applied this design pattern to our code. this is the SHA:
70cbb3df41642167306141b2a02185f2a116f164
Matcher.java 👍

Chain of Responsibility is a behavioral design pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

Application in `Matcher` class:

You can use this pattern to handle different stages of order processing, such as credit check, position check, and execution.

OrderHandler 👍

Chain of Responsibility allows a request to pass through a chain of handlers until it is processed.

Application: Implement different stages of order processing, such as validation, credit check, and execution, in a chain of handlers.

Security 👍

Purpose: Avoid coupling the sender of a request to its receiver by giving multiple objects a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Application in `Security` class: For processing orders, you can create a chain of responsibility to handle different types of orders.

3) Modifying Auction Matching

we implemented both auction and continuous states in Matcher class. A better implementation is to extract the auction-related functionality into a new class. This will help to separate the auction logic from the continuous matching logic, making the code more modular.
we can extract the common methods into a base class(`BaseMatcher`) that both `Matcher` and `AuctionMatcher` can inherit from.
In the `Matcher` class, we should remove the auction-related methods and make sure it focuses only on continuous matching.
Also, in the `OrderHandler` and `Security` classes, we should update the references to use the new `AuctionMatcher` class for auction-related methods.

4) Abstract Factory:

Abstract Factory is a creational design pattern that provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Matcher.java 👍

Application:

You can use Abstract Factory to create different types of `Order`, `Trade`, and `MatchResult` objects depending on the context (e.g., different market rules or order types).

OrderHandler 👍

Application: Use Abstract Factory to create different types of objects like `Order`, `Trade`, and `MatchResult` depending on the context (e.g., different market rules or order types).

Security 👍

5) Adapter:

Adapter is a structural design pattern that allows objects with incompatible interfaces to work together.

Matcher.java 👍

Application in `Matcher` class:

You can use an Adapter to integrate different order types or brokers that have different interfaces.

OrderHandler 👍

Adapter allows objects with incompatible interfaces to work together.

Application: Use Adapter to integrate different types of requests or brokers with varying interfaces.

Security 👍

Purpose: Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Application in `Security` class: If you need to integrate a third-party service that returns order information in a different format, an Adapter can be used.

6) Bridge

Bridge is a structural design pattern that lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other.

Matcher.java 👍

Application in `Matcher` class:

You can use the Bridge pattern to separate the abstraction of order matching from its implementation.

OrderHandler 👍

Bridge separates an abstraction from its implementation so that the two can vary independently.

Application: Separate the abstraction of order handling from its implementation, allowing for flexible and independent extension.

Security 👍

Purpose: Decouple an abstraction from its implementation so that the two can vary independently.

Application in `Security` class: If you need to support multiple matching algorithms, you can use the Bridge pattern.

7) Command

Command is a behavioral design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you parameterize methods with different requests, delay or queue a request's execution, and support undoable operations.

Matcher.java 👍

Application in `Matcher` class:

You can use the Command pattern to encapsulate a trading operation.

OrderHandler 👍

Command encapsulates a request as an object, thereby allowing for parameterization of clients with queues, requests, and operations.

Application: Encapsulate trading operations as commands to enable parameterization and support undoable operations.

Security 👍

Purpose: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Application in `Security` class: You can use the Command pattern to encapsulate operations such as placing, deleting, and updating orders.

8) Iterator

Iterator is a behavioral design pattern that lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree, etc.).

Matcher.java 👍

Application in `Matcher` class:

You can use the Iterator pattern to traverse through the trades.

OrderHandler 👍

Iterator provides a way to access the elements of a collection sequentially without exposing its underlying representation.

Application: Use Iterator to traverse through collections of trades or orders without exposing their underlying representations.

Security 👍

Purpose: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Application in `Security` class: To iterate over orders in the `OrderBook`, you can implement an iterator.

9) Mediator

Mediator is a behavioral design pattern that reduces chaotic dependencies between objects. The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object.

Matcher.java 👍

Application in `Matcher` class:

You can use the Mediator pattern to centralize communication between different components like orders, trades, and brokers.

OrderHandler 👍

Mediator reduces chaotic dependencies between objects by restricting direct communications and forcing them to collaborate only via a mediator object.

Application: Use Mediator to centralize communication between different components like orders, trades, and brokers.

Security 👍

Purpose: Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from

referring to each other explicitly and lets you vary their interaction independently.

Application in `Security` class: To manage communication between different services (e.g., matching service, validation service), you can use a mediator.

10) Builder

Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.

Matcher.java 👍

Application in `Matcher` class:

You can use the Builder pattern to construct complex `Trade` or `MatchResult` objects.

OrderHandler 👍

Builder constructs complex objects step by step, allowing for different types and representations of an object using the same construction code.

Application: Use Builder to construct complex `Trade` or `MatchResult` objects.

Security 👍

Purpose: Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Application in `Security` class: The `Security` class is already using the Builder pattern provided by Lombok. This pattern is useful for constructing `Security` objects with many optional parameters.